A Web-Based Framework for Personalised Language Rehabilitation

Roseanna Grundy **37158897**

SENG402 - Software Engineering Research Project

Supervised by Moffat Mathews

Submitted October 16th 2015

Department of Computer Science and Software Engineering
University of Canterbury

Abstract

This research project presents the research and development of web-based framework for simple language rehabilitation software that personalises to patient needs. This prototype uses a simple language task to demonstrate the viability of complete system that rehabilitates communication for people with Aphasia (PWA). Current rehabilitation techniques are repetitive and drill-based with little personalisation. Personalisation has been shown to enhance learning and maintain motivation in educational systems. We implement these same techniques in the application framework with the aim of enhancing quality of therapy and motivation in aphasic people. PWA can use the software independently and still receive a degree of personalised therapy without a human speech and language pathologist. Intelligent and adaptive software in this space does not currently exist. The framework uses Intelligent Tutoring System (ITS) techniques to achieve this goal. The ASPIRE ITS authoring tool was used to author a simple language tutor whose components would be integrated into the prototype framework. The domain ontology was constructed, task sets created, constraints generated, and feedback messages customised. This project found that the framework was viable, but that some functionality provided by ASPIRE may be insufficient. The components of the framework are designed to be modular, this way the constraints or feedback could be extended or replaced to meet future requirements.

Acknowledgements

There are a couple of acknowledgmens due to those who have helped me with my research, implementation and other project related work. Thank you to Dr Moffat Mathews, my supervisor. I wouldn't have got through this year without you. I also want to thank Dr Tanja Mitrovic and Jill de Jong for their help with the ASPIRE component of this project. Tanja's input on potential domains and drafting the ontology helped me evaluate the most suitable one. Jill helped solve my XML-RPC errors so that the framework could communicate with ASPIRE. Finally, acknowledging my Software Engineering 3rd Professional Year 2015 peers who have gone through the past three years with me, I wish you all the best.

Contents

Chapter 1

Introduction

Every day around 20 New Zealanders experience strokes. Stroke related health care needs are unmet for many New Zealanders. Most New Zealanders do not have access to appropriate, organised, inpatient stroke care [1]. It has been found that only 18 people need to receive this type of care to prevent one person from dying or being dependent at one year [2].

One third of the people who suffer a stroke will develop Aphasia. There are 16,000 New Zealanders living with Aphasia, this number increasing by 6 people every day [3]. Public awareness of Aphasia is lower than that for conditions with similar incidents and prevalence, such as Parkinson's disease [4].

1.1 Aims

As Aphasia varies in severity, and the ability to communicate is affected differently in each case, personalisation is essential [5]. Current computer-based rehabilitation does not make use of intelligent or adaptive systems. To improve the quality of therapy provided by computer-based rehabilitation systems, we translate components from Intelligent Tutoring Systems (ITS) to form an Intelligent Patient System that will utilise a patient model. This model can be used to record patient progress and identify strengths and weaknesses in communication and language areas. This report details a web-based framework as the basis for a personalised software solution. This framework employs a series of basic language tasks as a prototype to demonstrate that such a system is achievable and may be extensible.

1.2 Issues

Aphasia and treatment of brain injuries is a complex area to work in. It has been previously suggested that rehabilitation was not feasible for PWA. As such, exploration of therapy for PWA and much of the contributing research material is relatively new.

Existing software used in Aphasia therapy is often recommended to patients by a Speech and Language Pathologist (SLP). The software provides a selection of exercises spanning different language and communication components. Results from each session may be emailed to the SLP. However, this software lacks personalisation and relies on the SLP to devise a programme of exercises for the patient, as well as observe and analyse the data themselves. While moving away from manual data collection is helpful, the exercises themselves are uninteresting and repetitive.

Therapy is nuanced, and there are many aspects of Aphasia rehabilitation therapy that may not easily translate to a software solution. As such, simple language is the focus of the prototype. Even so, there may be subtleties involved in simple language that go unacknowledged due to our lack of domain experience in contrast to a qualified SLP. By making the framework modular, future augmentations may be able address this complexity.

The design for the system must be usable by those with varying degrees of physical impairment. Current ITS implementations are not designed to be used by those with cognitive impairments have a degree of complexity that would render them inaccessible and unusable by many PWA.

Much of the current software is designed for mobile and tablet devices, which are expensive. The risk of stroke is more prevalent in lower socio-economic areas [6]. Access to technology must be considered here.

1.3 Impact

As stated above, many New Zealanders do not receive satisfactory post-stroke care. In most cases, therapy should begin as soon as possible to ensure a patient has the best chance of recovery. [5]. By providing a personalised software solution we have the opportunity to improve the quality of therapy PWA receive, encouraging them to engage early and often with therapy material.

By developing software to support therapy, we are helping to increase awareness of Aphasia and the effects it has on those who live with or around aphasic people. Awareness can bring with it an increased ability to conduct further research in the area, i.e research grants.

Developing software to support SLPs will enable them to work with patients more effectively and efficiently. They will spend less time recording data, and will have more time to focus on the patient. Patients will be better supported and more motivated to work on exercises when computer based rehabilitation is personalised. Culturally relevant therapy is important [7], and the framework has the ability to facilitate this.

PWA are impacted socially and psychologically due to the communication barriers they experience. “Depression is a frequent reaction for both the person with Aphasia as well as for those close to them." [8] Early and continual rehabilitation has been shown to be effective. The current methods, especially repetitive and drill based exercises, are demotivating for patients. Improved personalised therapy has potential to emphasise rehabilitating early and encourage patients more effectively.

This project may elicit future research opportunities and developments. We have the ability to record patient data and make this available for further field research, after managing the privacy concerns surrounding patient data. Constructing a modular, extensible framework may provide a way to advance Aphasia research by encouraging software improvement.

Chapter 2

Background

2.1 Aphasia

Aphasia (also referred to as Dysphasia) is name given to a collection of acquired language disorders caused by damage to the brain, the most common cause being a stroke. “Language is the process in which thoughts and ideas become spoken. It involves the selection of words to be spoken, called semantics, and the formulation of appropriate sentences or phrases, called syntax." [9] Language is not exclusively about producing speech, but includes reading, writing and listening as well. Aphasic people may lose the ability to speak, read or write to varying severity [10]. Aphasia also affects visual language, such as sign language [11]. As the ability to communicate is compromised, Aphasic people experience isolation and depression. This negative emotional effect can have a great impact on the effectiveness and success of therapy [8].

2.2 Neuroplasticity

It was thought that once damaged, the brain could not be restored to it’s prior functioning. Neuroplasticity, also called brain plasticity, is an umbrella term for the process in which the brain’s neural synapses and pathways are altered due to the effect of environmental, behavioural and neural changes. It is the favoured theory to explain the brain’s ability to reorganise itself post injury.

Individuals with Aphasia have been observed to recover language function, research has implicated two forms of neuroplasticity [12]. Therapy has been found to induce neuroplasticity, and as functioning neuroimaging continues to improve, the effect of a particular therapy on the brain can be monitored and activated brain areas observed. [13, 14]

2.3 Intelligent Tutoring Systems and Personalised Learning

Computers have been used in education for over 20 years. Computer-based training (CBT) and computer aided instruction (CAI) were the first computer systems whose purpose was to teach [15]. Intelligent Tutoring Systems apply artificial intelligence techniques to education. They aim to model the behaviour of a good human tutor. In contrast to their less intelligent predecessors, these computer based tutors are designed to be an aid to human cognition. Instruction is tailored to the learner, and will adapt as a learner’s knowledge base grows.

ITS architecture is typically divided into four main components: the domain model, the student model, the communications module, and the pedagogical module [16, 17]. An expert model may be included [15].

The domain module represents domain knowledge in such a way that it can be interpreted and used by the system. Two main methods of modelling the domain are by using procedural rules within a Model Tracing approach [18] and using constraints within the Constraint-Based Modelling (CBM) approach [19]. Examples of MT tutors are Andes Physics Tutor [20], Practical Algebra Tutor (PAT) [21] and Geometry Tutor [22] while SQL-Tutor [23], EER-Tutor [24], Augmented Reality Tutor for Motherboard Assemby [25], and Furniture design Tutor [26] are examples of Constraint-Based tutors. CBM was pioneered at the University of Canterbury. In addition to the ontology of the domain, the domain module can contain other material such as problems or study notes.

The student model represents the current knowledge and skill level of the student and is used to make decisions within the tutor [27]. The quality and correctness of the student's interaction within the system is assessed to update the student model. The pedagogical module contains the pedagogical strategies used within the system. This module is responsible for decisions such as the problem a particular student should attempt next. The expert module is an expert's representation of the domain and is often runnable (i.e. it is capable of solving problems in the domain) [28].

2.4 Authoring Systems and ASPIRE

Authoring ITSs is a complex, time-consuming process that requires both ITS experts and domain experts to work collaboratively [29, 30]. Furthermore, often ITSs often developed from scratch.

To aid in this process, authoring systems have been created to automate a number of steps in the authoring process and to aid the non ITS expert in helping create a basic system. Current authoring systems do not do everything; they simply make the process much easier. ASPIRE[1] [29] is an authoring system developed at the University of Canterbury that automates several steps with building CBM tutors. ASPIRE also allows tutors to be deployed on its web server.

2.5 Intelligent Systems in Cognitive Rehabilitation

Intelligent Tutoring Systems teach a particular domain (such as Algebra, Kinematics) with the assumption that the student has no significant cognitive deficits as these cognitive aspects (such as memory, attention, executive functioning, and communication) are required for learning. Creating an intelligent system for rehabilitating cognitive deficits is therefore difficult as 1) it is difficult to model the particular domain of cognition (there are no clear rules or facts in the domain) and 2) it is difficult to ascertain whether an unexpected interaction is due to the student/patient's learning or other cognitive deficit.

Recently, some work has been done to incorporate techniques used for rehabilitation of cognitive deficits due to brain injury. The intelligent system for prospective memory rehabilitation was developed and trialled by the Intelligent Computer Tutoring Group at the University of Canterbury [31, 32]. We will be studying this and any other work conducted in this area during the literature research phase.

2.6 Computer Based Therapy

Computers have become increasingly used in Aphasia treatment over the past 20 years. This has led to research exploring the efficacy and effect of computer based treatment [33, 34]. Wertz and Katz asserted that clinicians cannot anticipate the behaviours of all patients, and computer based treatment programs only account for a number of contingencies. "As long as computer treatment programs are based on convergent, rather than divergent, theories of learning, computerised treatment will remain a subset of clinician-provided treatment" [33]. This statement highlights the necessity for computer based therapy systems to personalise and employ a range of therapy strategies. Guyard et al. [35] applied Artifical Intelligence in a domain specific way using a linguistic model. The framework described in this report can leverage Artifical Intelligence by way of ASPIRE's automatic constraint generation. This way a wider variety of domains could be supported with AI.

2.7 Relevant Software

Continuing from the discussion of prior research, this section explores the existing, relevant software that is designed to help PWA. This is a summary of the most popular providers of Aphasia software based on lists from the Aphasia Software Finder website [2]. Figure 2.1 on page 7 shows how different software compares to the proposed system.

	Cost	Feedback	Personalisation	Extensibile
Bungalow Software	$75-185	Partial	No	No
Tactus Therapy	$23-92	Yes	Partial	No
Parrot Software	$40 p/mth	Yes	Partial	Partial
React2	$50 p/mth	No	Partial	No
Smarty Ears	$12-23	No	Partial	No
Aphasia Therapy Online	Free	Partial	No	Partial
Proposed	Free	Yes	Yes	Yes

Figure 2.1: Existing software's features compared to proposed system. Feedback relates to the existence of cueing, Personalisation relates to whether the tutor adapts to the user, Extensible relates to the ability of the software to be extended or improved.

2.7.1 Evaluation of Existing Software Solutions

Aphasia Therapy Online

Platform: Browser
Cost: Free

Aphasia Therapy Online is free to use and does not require signing up. The system has region specific audio recordings, supporting accents from the UK, US and Australia. I have used this tutor several times to observe the range of drill tasks, type of cueing and degree of feedback. This tutor is aimed at adults, not children, which is evident from the muted green colour scheme - this is outlined as some tutors have interfaces targetting younger age groups and older users may feel condescended to.

Drill tasks are classified, which minimises the amount of information on screen. Tasks come under four main headings: Listening, Reading, Spelling, Naming. Any task may exist under more than one heading. Tasks are listed under Appendix B.1.

The existence of cueing (analogous to hints) varies task by task, but the system does provide some different cueing options. Simple cueing removes an incorrect multiple choice candidate after failure. In non listening tasks, the user may cue themselves with secondary audio by clicking the word or picture. Some tasks, such as 'Name pictures aloud' have a progression of available self-cues which bottom out at displaying the answer.

Bungalow Software

Platform: Windows
Cost: 69.50 AUD - 169.00 AUD

Bungalow Software does not offer trial versions of their software, but do have a detailed website. The photos on the website itself suggests that the software is quite dated. Bungalow market 23 different programs which are described in Appendix B.1. Most of these can be purchased in various editions - Standard, Deluxe and Professional. Examples of features available to Deluxe and Professional editions are memory and hint options (to further control the difficulty level), user progress tracking, and custom 'lessons' that therapists can build for patients. Playing audio to the patient appears to not be included by default in Bungalow's applications, and an 'Out Loud'

[1] http://aspire.cosc.canterbury.ac.nz
[2] http://www.aphasiasoftwarefinder.org/

version must be purchased to have this capability. The Standard editions of their software do not appear to include cueing options, recording of data, or personalisation.

Bungalow develops many different applications, each of which addresses a facet of speech and language rehabilitation. This fragmented approach to development and distribution suggests that each application is isolated. It is unlikely that data is shared between applications, and a therapist would be required to manually analyse and make conclusions across data sets.

React2

Platform: Browser and Windows (Flash based)
Cost: Subscription 20 GBP per month

React2 offers a two day trial of their software, which is Flash based and may be accessed online in the browser, or offline on a Windows PC.

React2 has five different sections: Auditory Processing, Visual Processing, Semantics, Memory/Sequencing and Life Skills. Each section has sub sections which further classify tasks. For example, Auditory Processing has the subsections Auditory Discrimination, Understanding Words, Understanding Sentences, and Understanding Conversations. Each task is described in simple English once a subsection is selected.

React2 has a clean interface, and the simple navigation means the user does not receive an excess of information in a single screen. Different task levels are available to control difficulty, but the numeric range of difficulty differs between tasks. It is unclear why one task only has three levels, where another may have eight.

In the full version of software, the site states that 'clinicians can set exercises'. React2 does not automatically construct lessons for the patient to complete, but like Bungalow Software a set of exercises can be created by the therapist.

Parrot Software

Platform: Browser, Mobile-friendly
Cost: Subscription 24.95 USD per month

I tried Parrot Software's system through their 7 day trial. Language selection offers Spanish and English, and the voice can be adjusted between Male and Female. Text-To-Speech (TTS) can be switched on or off.

Parrot Software have a unique approach to distributing their software. They have a wide array of tasks that they sell in application packages - similar to Bungalow Software. Parrot also makes every task available through an 'Internet Subscription'. This gives a user month by month access to the web app that lists all application package categories and subsequent tasks.

Parrot also uses a numeric system of ranking task difficulty. Each task has a number of lessons that imply an increasing difficulty. For each step within a lesson, independent of task type, only one incorrect answer is permitted. On the second incorrect answer, the system will display the correct answer and move on to the next task.

Parrot Software's system had a few unique aspects. One of these was congratulating the user after every complete step in a lesson, using TTS if it is turned on. I personally found this jarring, but was interested to see encouragement being employed in a system. Parrot Software also has tasks that use voice recognition and requires the user to speak into a microphone. When I tested this out I found it was accurate enough, but did not always interpret my speech correctly. I do not know how beneficial these tasks would be in practice, considering people with Aphasia may have impaired speaking abilities. Currently, speech recognition capabilities cannot compensate for impaired speech.

Smarty Ears Apps

Platform: iOS (Tablet)
Cost: 7.99 USD - 14.99 USD

Smarty Ears Apps do not offer any trial versions, but they do have video tutorials of some of their applications. I looked at two different applications from their Aphasia category.

- iName It - Helps with identifying and naming household items in several categories. Software does not assess correctness, that has to be done by the patient, speech and language pathologist, or caregiver.

- Reading Rehabilitation Toolkit - Word-Images match, Image-Words match, Phrase-Images match, Image-Phrases match, Read & Answer (read simple questions and answer based on the provided scenes). Has tracking, manual difficulty adjustment in the form of items/choices on screen.

Both of these applications are self contained, and only available on tablet. Interestingly, iName It was the only application I examined that approached therapy using a simulated 'real world' environment and strayed from the pure drill-based approach.

Tactus Therapy

Platform: iOS, Android (Tablet)
Cost: 14.99 USD - 59.99 USD

Tactus Therapy have developed around 15 mobile apps which are detailed in Appendix B. Most of these run on iOS, but some are also available on Android. Each app has built-in scoring that records use of cues, repetition, and errors. Uses 'Tests' to assess appropriate difficulty level, which can then be adjusted.

2.7.2 Evaluation Observations

There are many existing applications used for Aphasia therapy and rehabilitation, but many are separated into themed modules and used as supplementary tools where the planning and personalisation is done by a human speech and language pathologist. These tools can be expensive or inaccessible outside of a paid therapy session.

None of the main developers of computer based therapy software for PWA have systems that suggest exercises or tasks to patients based on recorded information. It is assumed that difficulty and patient needs will be assessed by an experienced human and tasks set based on this assessment. Some systems do make naive recommendations to the user this is either done naively in the form of a suggestion to raise difficulty level after a perfect score (i.e Parrot Software), or simply ordering exercises by difficulty. The feedback or cueing in applications is typically single level, and is not scaffolded.

Simple rehabilitation systems are overwhelming drill-based, but have many different categories of exercises for patients. For simple language, drills are repeatable and can be varied or extended without too much complexity. Only two other systems allowed for partial extensibility, that is, accommodated future exercise additions. Although it must be noted that building additional standalone systems for new exercises may equate to more money for the company.

Chapter 3

Approach

3.0.3 Delivery

Mobile, Desktop and Web delivery approaches were evaluated. A web delivery approach was selected due to high availability, low/negligible cost to access, and no installation requirements. I had advantages of prior web knowledge, and in theory the learning curve would be shallower.

A web application requires an Internet connection for access, but this access does not necessarily have to be maintained. For the purposes of this project, web appeared to be most suitable, and so was selected as the chosen delivery approach. Further detail of this analysis is included in the appendices.

3.0.4 Tools and Technologies

A variety of tools, frameworks and technologies were evaluated and used in the framework and are detailed below. Git, node.js, react.js were evaluated but do not feature in the existing framework. A description and analysis of these tools can be found in the appendices.

Languages

HTML, CSS and Javascript were used for front end development of the site. PHP was chosen over node.js for the web back end. These languages were used without any additional libraries or frameworks. This was a deliberate choice for several reasons. 1) to maintain a simple set of technologies and frameworks for the project, 2) to minimise overhead in establishing and learning a new framework, 3) to use familiar technologies that should be maintainable, as well as understandable for additional collaborators e.g future SENG402 students.

*AMP

*AMP is a generic acronym that represents the components of software bundles that are used to run dynamic websites or servers. There are many varieties, but the most popular variants (i.e LAMP, WAMP) are made up of four components. The first is the operating system that it is being run on - Windows, Linux, Mac OS, Solaris. Next, Apache - web server software. Finally, mySQL and PHP - database management system and the programming language.

Bitnami Stacks

Bitnami is an open source projects that provides many developer friendly tools. They produce open source installers, packages for web applications and development stacks. These stacks allow for easy installation of software and are pre-configured to work out of the box.

I used Bitnami's MAMP stack when developing on my laptop. It allowed me to run a local database and web server and quickly produce some example pages. As with a lot of pre-configured software, there are parts of the stack that I do not end up using, resulting in a bloated directory tree.

Virtual Machines (VMs)

The Intelligent Computer Tutoring Group (ICTG) has a number of tutoring systems that are hosted on the tutoring webserver (ictg.cosc) and the ASPIRE server (aspire.cosc). The tutors

usually communicate over various ports (e.g. ports 8000-8010, 8080, etc.) and are either deployed using the LISP webserver or Apache. These webservers are hosted on the local VM server farm.

ICTG have also been leading the rehabilitation research at CSSE (and now, the intelliHealth Systems Research Lab) with the prospective memory training for patients with brain injury. Even though we have looked into alternative means of delivering the rehabilitation system, we have decided to follow a similar delivery method as this has been used extensively and effectively so far. Once the VM is ready to be used by the public, it will be made available off campus (through the university firewall).

Although VMs on the CSSE server farm are not backed up, the MySQL central database that contains the data is backed up. Our current development methods also mean that all code is on the development machines as well as on the VM.

The production VM has been set up and configured with a non default password. The base page has been modified, but no code has been pushed from development to production at this stage.

Pure.css

Pure.css is based on Normalise.css, and provides cross-browser layout and styling for native HTML elements. Pure aims to be responsive, and handle different screen sizes and platforms.

HTML is not rendered identically across all browsers, and there are multiple methods to address this. CSS resets have previously been a way to address this, as they reset all browser initialised CSS. Solutions such as Normalise.css are seen as more sophisticated as they render websites consistently across different browsers. This requires continued use of classes provided by the style sheet. Pure.css has been used to create mobile compatible pages, address browser differences, style elements easily, and allow for grid positioning.

PHPUnit

PHPUnit was used as a unit framework to test my PHP code. Development is hosted on GitHub by Sebastian Bergmann. It was released in 2004 and is still being updated. I used it for simple assertions to confirm calls were reaching ASPIRE and my PHP was processing the XML appropriately. One of the most useful aspects of PHPUnit that I found was the @depends annotation which allows you to describe what tests in the suite depend on other tests passing. This is extremely informative for developers, as well as improving the accuracy of test output.

Composer

Composer is a tool for dependency management in PHP. Dependencies are declared in a .json file and Composer will manage (install/update) these when the user requests particular actions via the command line. Composer was used to install and update PHPUnit 4.8 (a prior release as I was working with PHP 5.5.29 instead of 5.6.

While Composer created an additional folder for dependencies, of which I was only installing one, the configuration options allow you to have more control over a project's PHP dependencies and I preferred installing PHPunit via Composer rather than manually.

Chapter 4

System Overview

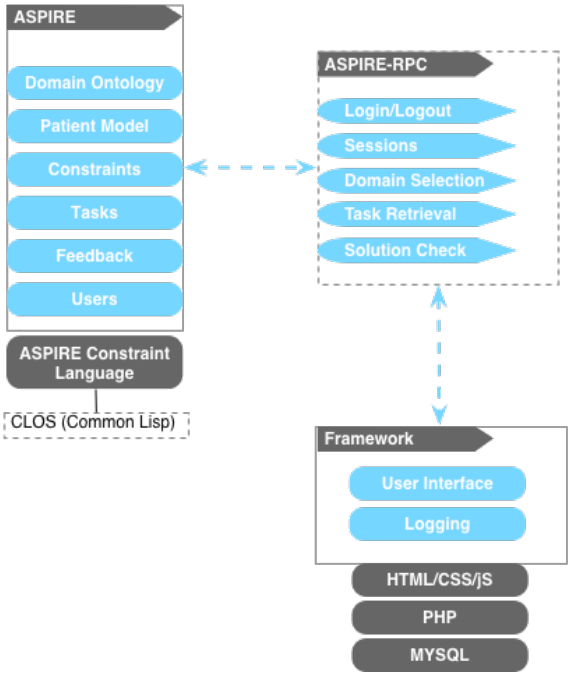


Figure 4.1: Current structure of the prototype system.

Figure 4.1 shows the current system overview, with ASPIRE-RPC being the communication channel between the framework and ASPIRE. For this prototype, all content generation is performed on ASPIRE. Modules can be managed within the framework component, but ASPIRE stored data must be retrieved. In future, modules from ASPIRE could be replaced by our own, or be processed by an additional system layer - such as adding appropriate audio to feedback messages.

4.1 ASPIRE

ASPIRE allows accelerated authoring of a CBM ITS. For this project, ASPIRE is used to host the patient models for each patient, the therapies, the domain ontology defining task structure, and the task sets. Session management, users, feedback and problem definitions are provided by ASPIRE and are requested over ASPIRE-RPC. ASPIRE provides the foundation for creating tasks and associated components that could be integrated into a web framework. My aim was to develop an example task for the project prototype. I completed, deployed and demoed my domain at the end of Term 2.

4.1.1 Research

The aim was to convert an exercise suitable for PWA into an ASPIRE task. After identifying the task, I had to create a suitable domain ontology to represent the task components.

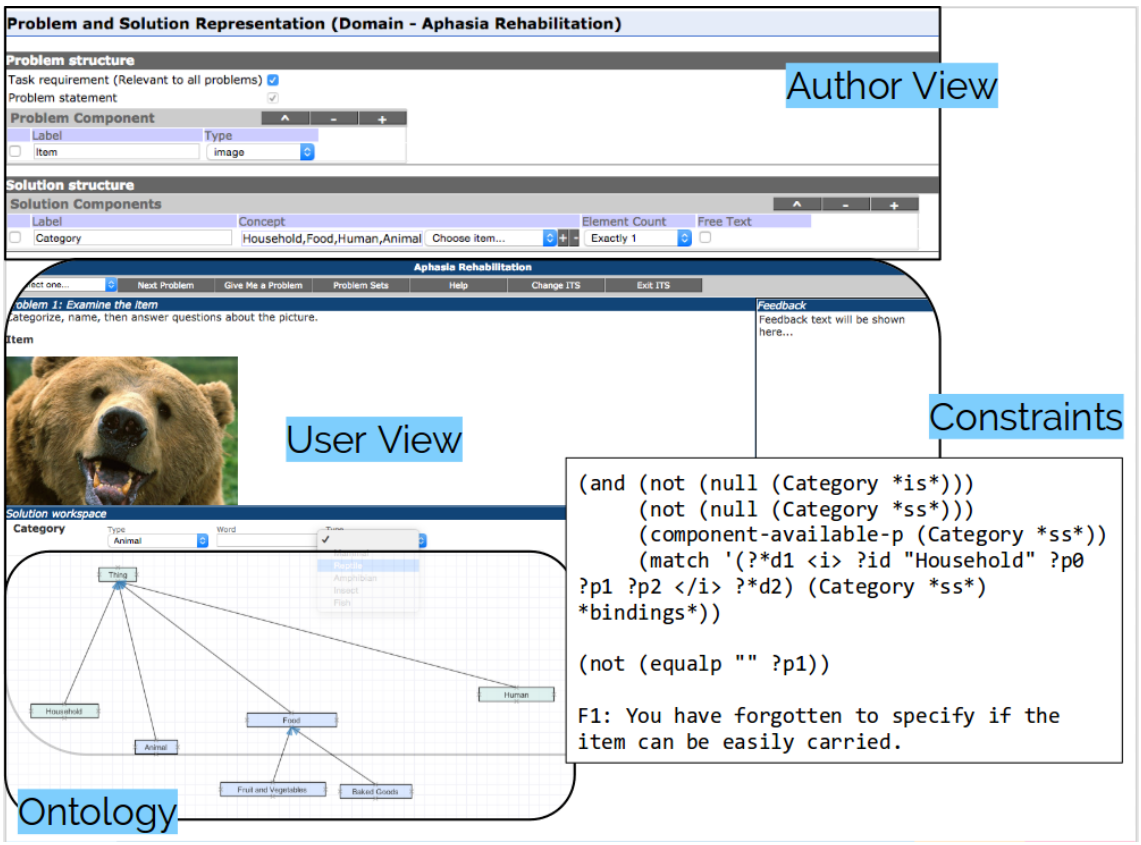


Figure 4.2: Overview of the ASPIRE components of the system.

The exercise had to address simple language, and be suitable as a task on the computer. The exercise must have a way or ways of increasing difficulty. Firstly, I examined a range of different activities used in Aphasia therapy and classified them in different ways. Some questions I identified for classification:

- What core skill is being stimulated?
  - Listening
  - Speaking
  - Reading
  - Writing
- What action will the patient perform?
  - Matching (like object for like object)
  - Identifying (like matching but with different object representations e.g text to pic)
  - Completion (e.g sentence, cloze activities, image group)
  - Comprehension (understanding)
- What language structures are being used?
  - Letters
  - Words
  - Phrases
  - Sentences
  - Paragraphs
  - Conversations
- What is the stimulus/stimuli?
  - Auditory/Music
  - Auditory/Speech
  - Visual/Text
  - Visual/Pictures

One of the problems was that most Aphasia drills are very simple. Typically, they involve one step and the require a singular user input before a result is shown. Answers are usually shown as multiple choice with one correct answer. This doesn't sound like a problem at first, but ASPIRE expects and works best when tasks have multiple steps and multiple answer components. Minimal steps and user inputs limits the sophistication of feedback that could be provided to the patient. ASPIRE-only feedback is satisfactory for this project as the focus is on the framework and not module development.

I examined three different candidate therapy exercises which at a glance seemed to be viable

options to use with ASPIRE. I looked at Categorisation, Naming and Semantic Feature Analysis. I met with Tanja Mitrovic to discuss how these could be implemented with sufficient complexity. I attempted to implement a Naming task in ASPIRE, detailed in appendix D.1. After failing to achieve this I settled on Semantic Feature Analysis.

4.1.2 Semantic Feature Analysis

Semantic Feature Analysis (SFA) was suitably appropriate and versatile. This technique involves naming a target, which is prompted by a picture stimulus. In addition to word retrieval, a series of questions about the semantic features are asked. This requires comprehension and identification skills. For example, the picture stimulus may be an apple, features could be the colour (red), where it is found (tree) and so on. SFA has been demonstrated to be effective in rehabilitation, in particular it was used as part of a brief, intensive therapy study and was observed to induce neuroplasticity [13]. SFA can be scaled in complexity by using more abstract features. This was defined in ASPIRE as a non procedural task.

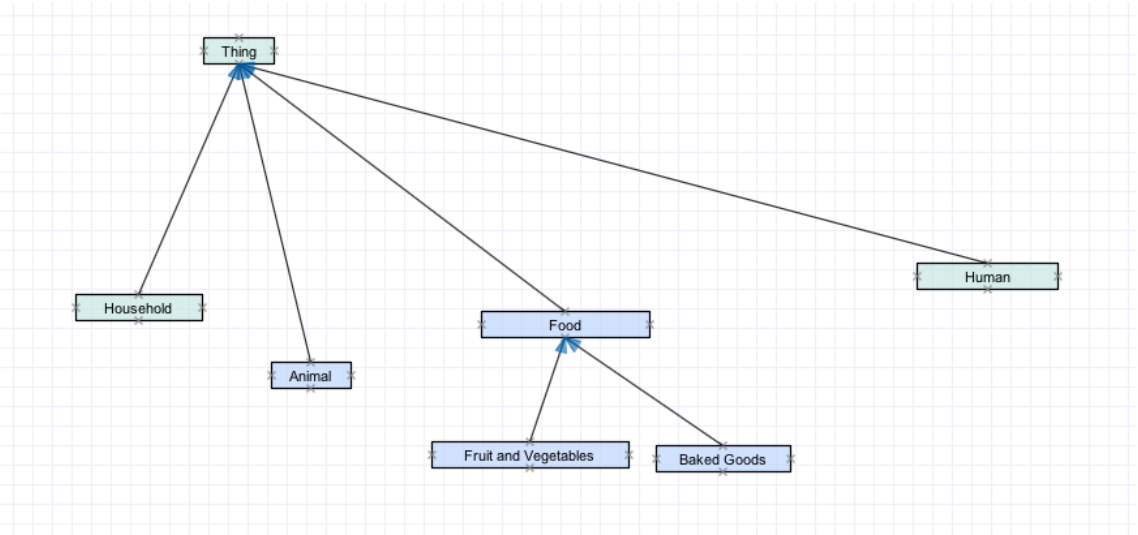


Figure 4.3: The Semantic Feature Analysis Domain Ontology.

4.1.3 Ontology

The domain ontology defines how problems are presented and completed. The ontology for SFA is quite simple, pictured in Figure 4.3 on page 14. Each specialisation represents a different set of problems in the domain, with problem sets existing for 'Animal' and 'Food/Fruits and Vegetables'. Semantic features are listed as attributes as shown in Figure 4.4

Name	Type
P Word	String
P Colour	Symbol
P Type	Symbol
P Is it healthy?	Boolean
P Do you peel it?	Boolean
P Has seeds?	Boolean
P Has a pit?	Boolean

Figure 4.4: Feature Attributes for 'Fruits and Vegetables'

SFA was selected primarily because it had sufficient complexity for ASPIRE, mapped well to a domain ontology and included categorisation and naming components similar to that of my initial therapy exercise ideas.

4.1.4 Problem Structure

The problem structure requires the object category to be identified, named and the enumerated semantic features answered. Figure 4.5 (pg. 15) shows the problem statement is an image, and there is a single solution component 'Category'. Once this is selected, features are enumerated (Figure 4.6 (pg. 15).

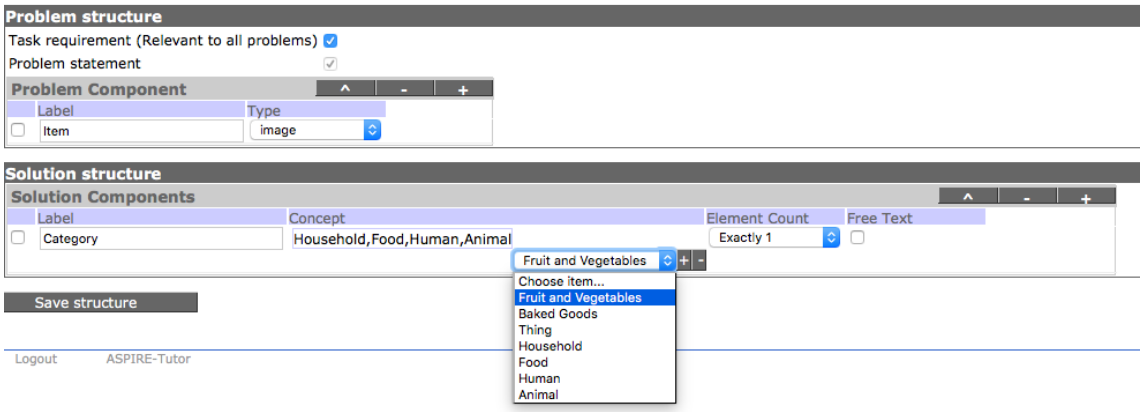


Figure 4.5: Semantic Feature Analysis Problem Structure

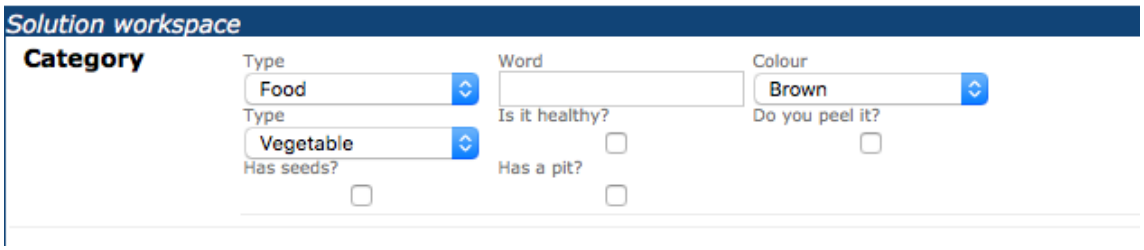


Figure 4.6: Semantic Feature Analysis Interface Example

4.1.5 Constraints

The ASPIRE Constraint Language is based on the Common Lisp Object System (CLOS). Constraints can be generated automatically from a set of provided answers and input requirements. In total 14 syntax constraints and 8 semantic constraints were generated. Figure 4.7 (below) shows one of the syntax constraints in ASPIRE.

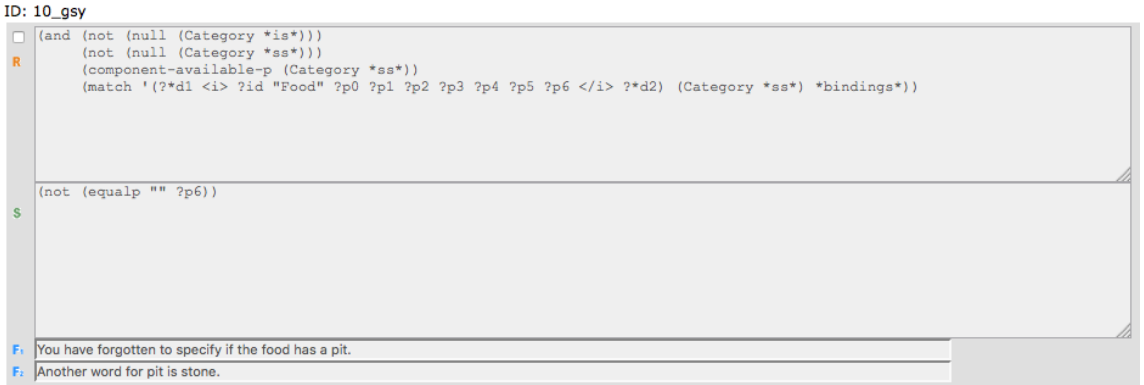


Figure 4.7: Example Syntax Constraint

My attention here was directed to formatting the feedback messages as in their raw state they are unhelpful for patients. There are two levels are feedback for each constraint. Syntax constraint messages alerted the patient to untouched input (Feedback level 1), and cued them on the input (Feedback level 2). Semantic constraint messages alerted the patient to mistakes (Feedback level

1), and directed them how to fix their mistake (Feedback level 2). I also included "encouragements" within the feedback messages, such as praising for correct category choice.

4.2 Framework

4.2.1 Communicating with ASPIRE

The core part of the framework is the PHP interface to ASPIRE-RPC. For every method call described in the ASPIRE-RPC documentation (Appendix D.2), there should be an equivalent PHP function. It is envisioned that when framework modules are developed, they will utilise the calls they need and process the responses.

As ASPIRE-RPC expects to receive XML and responds with XML, PHP's SimpleXMLElement objects are used. I found XML and the XMLElement to be difficult to work with. XML requests have to be constructed very carefully, and are sensitive to spaces. Incorrectly positioned spaces led to malformed XML, breaking calls.

Parsing the XML successfully confirmed that components from ASPIRE could be accessed and displayed in the front end. An example login call is shown in Figure 4.8. The PHP interface sets up the request using cURL (Figure 4.9, pg 17), formats the XML to send (Figure 4.10, pg 17).

```php
/**
  * Returns SimpleXMLElement <response> with a 'status' child element.
  * On success, status = 0, with a 'session-id' integer child element.
  * On failure, status = -1, with a 'errMessage' string child element.
  *
  * Type coercion is needed when comparing child element values.
  */
function login($username, $password, $affiliation) {
  $params = array(
  'username' => $username,
  'password' => $password,
  'affiliation' => $affiliation);

  $method = 'aspire.login';

  $data = construct_data($method, $params);
  $output = get_curl_response($data);
  $xml = extract_nested_xml($output);

  return $xml;
}
```

Figure 4.8: ASPIRE login function in PHP

4.2.2 User Interface

A series of interface wireframes went through on-paper and on-whiteboard iterations. The interface had to be easy to comprehend and navigate. Attention was directed to the number of items on screen at once, as PWA may have a lower threshold for their cognitive load. Figure 4.11 shows roughly what was envisioned for the front end component of the site. This is implemented using a pure.css template with example data to mimic the results of the ASPIRE PHP API.

```php
function get_curl_response($data) {
  $url = 'http://aspire.cosc.canterbury.ac.nz:8001/rpc';
  // creating the curl handler
  $ch = curl_init();

  // set options
  curl_setopt($ch, CURLOPT_HTTP_VERSION, 1.0);
  curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0); // ssl stuff
  curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0); // vulnerable, man in the middle attack
  curl_setopt($ch, CURLOPT_POST, 1);
  curl_setopt($ch, CURLOPT_URL, $url);
  curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-type: application/xml'));
  curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

  // get return response
  $output = curl_exec($ch);

  // close the curl handler
  curl_close($ch);

  return $output;
}
```

Figure 4.9: Setting up HTTP POST request with cURL

```php
function construct_data($method, $params) {
  $param_html = "";

  foreach ($params as $key => $value) {
    // The appended string must have zero spaces, or the request breaks.
    $param_html .= "<$key>$value</$key>";
  }

  return "<methodCall>
  <methodName>$method</methodName>
  <params>
  <param>
  <value><string><![CDATA[" . $param_html . "]]></string></value>
  </param>
  </params>
  </methodCall>";
}
```

Figure 4.10: Translating PHP array into XML as a string to send in request.

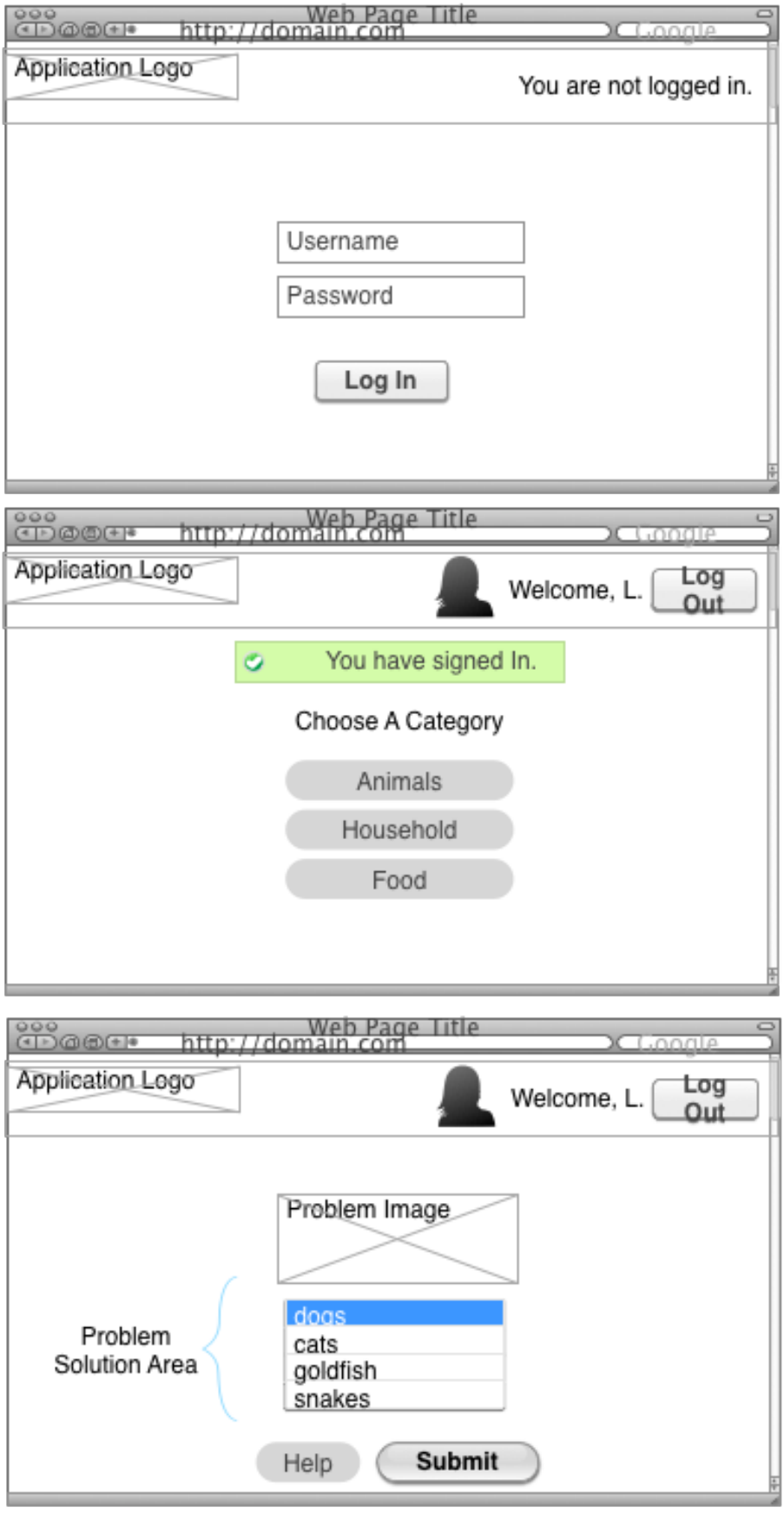


Figure 4.11: Wireframes showing proposed simplicity of the web interface

4.3 Testing

4.3.1 Initial Research

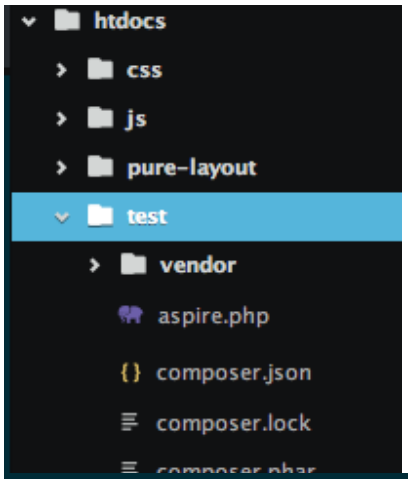
Initially the project was presumed to have a major Javascript component, and so research into Javascript testing was conducted. Potential candidates for testing frameworks/libraries included should.js, js-must, unit.js, mocha.js and chai.js (which run on node.js), Jasmine. During developed, I planned to use an ad hoc testing framework using the console as input, such a method is explained in an article from Smashing Magazine "Introduction to Javascript Unit Testing"[1]. Through SENG365 and SENG301 I had varying theorectical knowledge about protractor, Cucumber, Selenium, SimpleTest and PHPUnit.

4.3.2 Unit Testing

One of the most important areas to test was the ASPIRE-RPC PHP interface. My initial framework is dependent on these calls to authenticate users, obtain sessions, select the desired domain or subdomain and retrieve problems.

Initially these were tested using a webpage with echo statements and conditionals. One of the issues with echo statements is that type coercion happens, but is often concealed. It was useful to colour format the output and display it on a webpage, but ultimately it was not efficient to rerun, assertions were weak and tests were not self documenting.

Translating tests to PHPUnit exposed a problem with SimpleXMLElement objects as it does not allow for type coercion unless explicitly called. 15 tests were authored to test the communication to and from ASPIRE, using PHPUnit 4.8.12. PHPUnit was installed in the test directory using a PHP depency manager, Composer. Figure 4.13, page 19 shows the converted tests. These are so much cleaner than what I had originally which involved wrapping each method in a test function, then printing the output to screen.




```php
/**
 * @depends test_login_success
 */
public function test_logout_success() {
  $login = login("pwa01", "potato", "ictg");
  $xml = logout($login->{'session-id'});
  $this->assertEquals(0, (int) $xml->status);
}

public function test_logout_malformed() {
  $xml = logout("365025348424363795508227096069617442945350208963");
  $this->assertEquals(-1, (int) $xml->status);
}

public function test_logout_oldsession() {
  $xml = logout("3650253484243637955082270960696174429453502089633");
  $this->assertEquals(-1, (int) $xml->status);
}
```

Figure 4.13: Example unit tests using PHPUnit.

[1] http://www.smashingmagazine.com/2012/06/27/introduction-to-javascript-unit-testing/

4.3.3 Acceptance Testing

Acceptance testing was performed on a manual, ad hoc basis between student and supervisor throughout the course. This was not manually documented, and such documentation may have reflected many changes in requirements. As project scope was refined and changed, the ability to effectively unit test, test system integrations and adequately assess the requirements and goals for the system became harder.

Chapter 5

Discussion

5.1 ASPIRE

Most computer aided Aphasia therapy exercises are extremely simplistic and provide no sophisticated feedback. Exercises that are more complex are evaluated by what is closest to a Model Tracing approach. This means that when evaluating a patient's solution for correctness, the steps taken to reach that answer are matched to one or more correct sequences of steps. To give sophisticated feedback for Naming exercises, model tracing would allow for problem specific cues and prompting. Unfortunately, this would mean complex and time intensive process of constructing many models to account for multiple scenarios.

Another difficulty related to the domain's simplicity was that ASPIRE is reasonably verbose in terms of user interface and feedback. The interface itself would most likely prove too busy and complex for an aphasic person to interact with.

I resorted to generic hints and cueing when developing within ASPIRE. The use of variables or access to question sensitive information in constraint violation text would make feedback more sophisticated and useful, which is potentially something that could be added in the framework.

ASPIRE has lack of customisation in regards to problem structure layout and user interface. This customisation is intended to be achieved through developing the web interface while using the ontology and content on ASPIRE using remote procedure calls.

5.2 Framework

ASPIRE-RPC call examples in the manual are given in Lisp form. Interacting with ASPIRE using PHP was not the same. Working with PHP adapted calls to ASPIRE outlined current issues with the ASPIRE API. While one of these was fixed, not all of these are able to be addressed in the time available. A failing test has been written for setting the subdomain (selecting the task set). At the time of writing it responds with a "server-error" message. Due to another system's dependence on the current configuration of ASPIRE, this was not able to be fixed without compromising the other.

Despite an early design decision not to use a framework to keep the project lightweight, in hindsight I should've researched frameworks further - including speaking to more experienced PHP developers. A framework potentially could have provided some of the basic login functionality, user permissions and logging. I had only worked with CodeIgniter previously, which was not a good fit given my requirements. The state of the project still allows for a transition to a PHP framework with little upfront cost.

Much care and thought went into how the ASPIRE-PHP API would be structured. These functions are well documented, and intend to match the prior API closely for familiarity purposes, with most responses being in PHP's SimpleXMLElement. This did not make sense in all cases, especially when there were an arbitrary series of child elements in the response. In this case, an associative array was returned. If I was to continue working on this project I would adapt all calls to return associative arrays as they are easier and more pleasant to work with in PHP than SimpleXMLElements which require value casting.

5.3 User Interface

User interface testing was beyond the scope of this Honours project, but there is sufficient material in existence looking at interfaces designed for PWA. Research could be conducted on different screen configurations, tracking head and eye movement to assess what interface is most beneficial to patients.

5.4 Conclusion

This project established that it was possible to create a set of tasks in a valid domain and develop the beginnings of a web framework, capable of displaying the series of tasks. It is not known, but assumed that once ASPIRE is fixed to allow subdomain and problem calls, they will be able to be parsed and displayed using PHP. Especially successful here was the research component, which surveyed all major providers of software and confirmed the frameworks proposed goal was unique.

This solution could be extended in multiple ways, and one of the project's distinct aims is to make the framework open for extension. It became apparent through development that the existing level of feedback would not be practically sufficient for real world PWA, and is just one of the system components that further research would benefit from.

Appendix A

Milestones (Proposal)

Date	Description
23/03	Proposal Due
27/03	Examine current rehabilitation for simple language.
03/04	Explore existing rehabilitation software.
10/04	Research potential technical and architectural solutions.
17/04	Design a preliminary software solution.
24/04	Begin implementation phase within a Virtual Machine (or equivalent).
08/05	Progress Report #1 Due.
18/05	Explore sources of domain content, look at generation, multi media.
29/05	Create ontology and constraints in ASPIRE.
03/06	COSC420 Assignment #3 Due, of which ITS development contributes to the project.
26/06	Integrate ontology and constraints with VM (or equivalent) implementation.
29/06	Begin content generation
17/07	Progress Report #2 Due
20/07	Begin testing phase
14/08	System behaviour evaluated by a Communication Disorders and SE experts.
21/08	Data analyses
09-10/09	2015 Postgraduate conference and project presentation.
18/09	Modifying the program according to feedback from the expert trials
02/10	Final report writing and preparing for practical demonstration.
17/10	Final report due. Practical inspection/demonstration due.

Appendix B

Current Software

B.1 Tasks or Programs

B.1.1 Aphasia Therapy Online

- Listen and choose picture
- Listen and choose word
- Listen and choose letter - uses different fonts.
- Type a spoken word
- Type the name of a picture
- Read and choose picture
- Read and choose word
- Match capital and lowercase
- Read and answer yes and no questions
- Name pictures aloud

B.1.2 Bungalow Software

- Aphasia Tutor 1: Words - Learning letters & words. Letter and word level reading comprehension. Written naming. Speaks the letters and words aloud in a human voice for extra help (Out Loud version).
- Aphasia Tutor 2: Sentences - Sentence completion and word retrieval. Sentence level reading comprehension. Reads the sentences aloud in a human voice for extra help.
- Aphasia Tutor 3: Story Reading - Paragraph and story level reading.
- Aphasia Tutor 4: Functional Reading (recipes, T.V. guide, schedules, etc.)
- Sights'n Sounds 1 - Verbal naming (spoken word retrieval) and oral reading.
- Sights'n Sounds 2 - For speech production and articulation at the sentence level. Records patient's voice and plays it back along with a model sentence for comparison.
- Numbers'n Sounds - Reading numbers aloud.
- Synonyms, Antonyms, & Homonyms - Matching and written word retrieval.
- Sentence Shaper 2 - Helps patients turn halting, hesitant speech into conversational phrases and speeches.
- Categories and Words - Word and phrase level reading comprehension. Patient must choose which word does or does not belong with the others.

- Understanding Questions - sentence-level reading comprehension. Comprehension of spoken questions (Out Loud version).
- Direction Following - Following written directions at the phrase level up to the paragraph level. Comprehending spoken instructions from phrase level to 3 step paragraph long instructions (Out Loud version).

B.1.3 Tactus Therapy

- Language TherAppy - Targets all four language modalities—reading, writing, speaking, and listening (combines 4 separate *Appys)
- Question Therapy - Targets both the receptive and the expressive use of questions, concentrating on yes-or-no questions and the who, what, when, where, how, and why of a situation.
- Asking Therapy - provides four unique activities to help users understand and use proper verb tenses for yes/no questions and select the correct question-starting word for Wh-questions.
- Answering Therapy - helps people with impaired communication skills practice responding to yes/no and Wh questions.
- Reading TherAppy - Users can progress from reading word to reading stories using the comprehension tasks in this app: Phrase Matching, Sentence Matching, Phrase Completion, and Sentence Completion.
- Naming TherAppy - Naming Practice, Describe (Semantic Feature Analysis for nouns and Phonological Components Analysis for all words) and Flashcards (choose your own approach). Built in cueing hierarchy.
- Comprehension TherAppy - provides three user-friendly modes—Listen, Read, and Listen & Read—for unlimited practice with full-colour photographs and real recorded voice.
- Category Therapy - group items and name common characteristics. Four exercises: Find, Classify, Exclude, and Add One. Three levels of difficulty: Concrete, Subcategories, and Abstract.

Appendix C

Approach and Technology Evaluation

C.1 Delivery Approach

C.1.1 Mobile Application

Mobile applications are popular in the therapy space, especially tablets as they have a large screen, they are portable, tactile and can be operated by those with accessibility requirements. When therapy systems are accessed and used frequently, the benefit a patient will receive increases. A mobile could facilitate this repeated and ease of access. Mobile applications require simple interfaces, and this constraint benefits those who processing capabilities may be limited.

C.1.2 Desktop Application

Making use of previous experience I could have developed a desktop application using Java. I would have resources within my Software Engineering cohort and also of personal experience. Java is widely used, is well discussed and documented online, and the resulting application would run on multiple machines using the JVM.

C.1.3 Web Application

The system could be developed on the web, with the advantage of a website being easily accessible from both mobile and desktop browsers. Responsive design allows touch sensitive and mobile compatible devices to be built. Multiple form factors will require different user interface designs, as what makes sense on a mobile device does not always translate to being effective in the browser.

C.1.4 Analysis

When beginning this project, I had no former experience in developing mobile applications. The learning curve for a mobile approach would have been steeper, especially developing for native applications like Windows Phone or iOS.

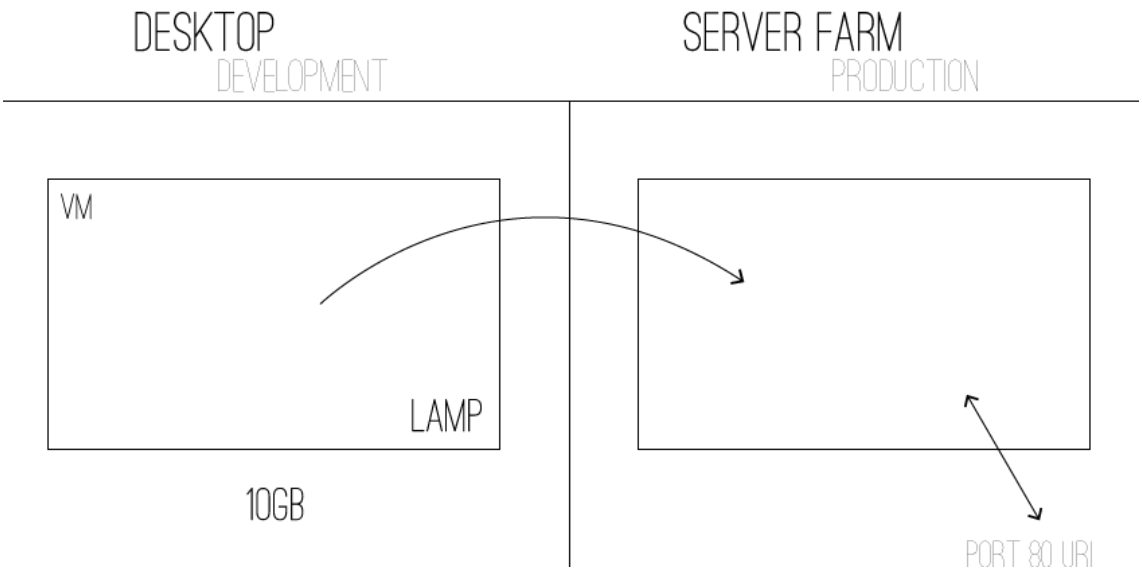
A desktop application requires a user to go through some installation process before the program can be run. Not all users are comfortable with this. If a user has multiple computers they interact with, this would require multiple installations. Deployment of the application may prove costly and delivery itself is restrictive.

While there are some desktop therapy applications, a lot of these appear to be outdated, or have moved towards providing the same application in a web app form.

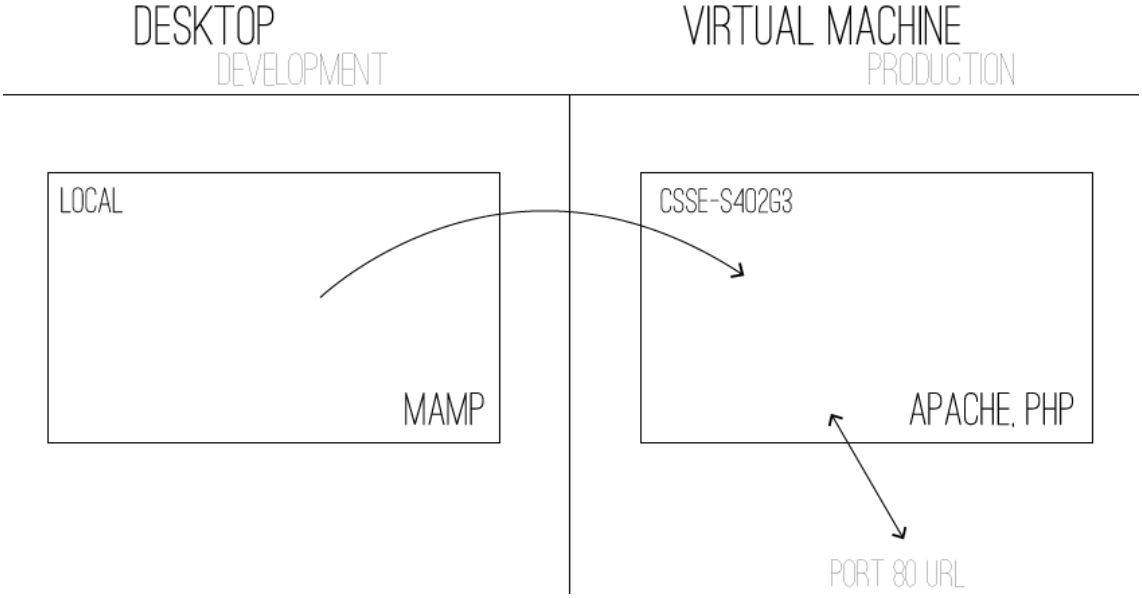
Web applications can be extended to mobile and do not have the set up cost of a desktop application. For this project it is most suitable to develop a framework in.

C.1.5 Proposed Approach

The first approach was to develop on a Linux machine at the University of Canterbury on the campus network. A virtual machine would be set up with a LAMP stack installed. The project code would be pushed to a production virtual machine in the COSC server farm which could then be made externally accessible.



The second and current approach is to develop locally on my own laptop that runs OS X. This means development is not restricted to being on site or transferring files back and forth. The laptop will identify a local folder with MAMP stack installed, and manages a web server and database through Bitnami's bundled management application. The production virtual machine was set up with a unique web address. This VM is without an installation of mySQL. The COSC central mySQL database will be used instead if necessary, as it is backed up.



C.2 Tools

Git

Git is a Version Control System (VCS) that is widely used in all sizes of Software Engineering projects. I gained experience using Git as a VCS for the group project in SENG202. I have since used git in a commercial setting during internships. Git is a potential way to manage and track changes as I develop my project and associated documentation. At this stage in the project I am utilising other methods of sharing documents with my supervisor, such as Google Drive, which itself has a rudimentary method of tracking changes. At this point the complexity of the related project code and need to expose it for collaboration has not been sufficient to use Git. Later I may utilise Git, available through GitLab[1].

[1] https://eng-git.canterbury.ac.nz/

Node.js

Node.js, also known as Node, is an open source, cross-platform I/0 environment built on Google Chrome's Javascript runtime. Node is used to build concurrent web applications that are scalable and high performing. It has a asynchronous, event-driven I/O model, allowing JavaScript developers to work server-side.

From my research I have discovered that Node has been widely, and often criticised for being overused and over hyped. I had attended a recent conference centring around Node, called NodeConf. Despite this, I did not have an accurate picture of what Node was intended to be used for. For my project, node.js may be an alternative to PHP for working server side. To date I have worked through five NodeSchool[2] 'learn you node' tutorials.

React.js

React is used for building user interfaces, it is a library developed by Facebook and Instagram. React uses a fast, internal mock DOM to perform diffs and computes the most efficient DOM mutation. It does not modify the DOM unless it needs to. Manipulation of DOM can lead to a lot of inefficiencies in web applications, so a way of managing this is important. React components are like JavaScript functions that have properties and state, and render to HTML. Managing the interface and the relationship between view and model is important, especially as both are liable to change many times during the project.

Audio APIs

Audio will feature in the project, so I looked at the related audio APIs that are available for use, to see if any could be used and how.

- Web Audio API - Provides audio processing and synthesizing - offers things such as linear effects, spatialised audio, biquad filtering. Not immediately relevant but may prove interesting to process sound in ways that affect the difficulty of an audio based task.
- Web Speech API - Is not officially supported, and is not supported fully in any browser. Web Speech encompasses speech recognition and voice synthesis. There is partial support in both Google Chrome and Safari - both support voice synthesis (Text-To-Speech). Speech recognition works in Google Chrome, but is not accurate enough to be currently worth an investment in. I am considering the benefit of using TTS. Early on, it could mean I could generate some initial soundbites that would not require recording, uploading and embedding. Later it may provide a layer of accessibility.
- Microsoft's Project Oxford - A selection of Machine Learning APIs very recently released (30th April). Despite being tied to Microsoft, the Project Oxford Speech API may prove more sophisticated than Web Speech. Documentation for the REST API is available online, but I have not investigated further at this point.

[2] http://nodeschool.io/

Appendix D

ASPIRE

D.1 Progress - ASPIRE

I was unfamiliar with ASPIRE at this stage, and one of the best ways to figure out what is possible is to begin developing example tutors. Meeting Tanja was helpful in terms of exploring potential approaches, seeing demos of previously implemented domains and viewing the variables behind associated problems. Unfortunately, because I still had limited knowledge of what ASPIRE could do and how to use it with my domain I was still unsure about the suitability of each candidate therapy exercise.

Initially I was looking to create problems that target anomic Aphasia, as many existing drills target. The focus for these drills is often choosing the name of a presented picture, picking a word from a limited series of options. Naming exercises proved difficult to implement as I desired, despite the simplistic nature of the problem independent of ASPIRE. As I was building the ontology and the problem structure, I realised that this therapy exercise did not work well with ASPIRE. Multi choice selection is achievable in ASPIRE, but is only really viable for a small number of values - too many values in a dropdown is poor from a user experience perspective, and the excess of words is likely to dramatically hinder an aphasic person.

Assuming a tutor has a large dictionary of possible words, only a subset of these words should be displayed for multi choice selection. In most external drill examples, one answer is among 3-5 potential words. This can be achieved in a roundabout way by explicitly defining the words to choose between in the problem statement. Each word maps to a generic option in a drop down. i.e A = word1, B = word2, C = word3. Select A, B or C.

Another way around was avoiding multi choice altogether and implementing naming problems with the user typing in a textbox. This simplified things, yet still reflected existing naming drills.

I abandoned Naming exercise attempts due to the difficulty with providing valuable feedback and developed two Semantic Feature Analysis problem sets as the ontology mapped well to the problem structure in ASPIRE.

D.2 ASPIRE-RPC

ASPIRE's API is detailed on the following page.

ASPIRE-RPC

Remote Procedure Calls allow for applications to utilise the Tutoring capabilities of ASPIRE. This allows application developers more control over how information is displayed to the user. RPCs allow developers to create a custom interface (ie not a web browser), and to integrate a Tutor into their application. This will only be used for tutoring, not for authoring. The RPCs should support the usual tutor functions, such as Login, Select Problem, Submit Problem etc.

RPCs can be accessed through the url http://server:8002/rpc, where server is the address of the ASPIRE-Tutor server.

When normally using ASPIRE, through a browser, much information is stored in the session. This has to be replicated in some way for RPCs. The most likely solution is to do what Allegro Server does when a browser does not support cookies: a session id is generated and must be added to the URL string for each call. This would allow all the existing logic in ASPIRE to be used, which significantly reduces development effort, both for the initial RPC work and perhaps more importantly in maintenance. The login method returns a session-id on a successful login, which must be used as a parameter to the other methods.

Note: There is no checking performed on the input strings, so they need to be correct.

The API is based on the WETAS RPC API.

API:

aspire.login

What it does: logs a user in

Input:

- <username>username</username>
- <password>password</password>
- <affiliation>affiliation</affiliation>

Returns, on successful login:

- <status>0 (user logged in)</status>
- <session-id>session-id</session-id>
  and on unsuccessful login:
- <status>-1 (error)</status>
- <errMessage>text error message to be displayed</errMessage>

At this stage the password is sent in plain text

aspire.logout

What it does: logs the user out.

Inputs:

- <session-id>session-id</session-id>

Returns:

- <status>0 (OK) or â€“1 (error)</status>

- <errMessage>error message (if status=error)</errMessage>

aspire.listDomains

What it does: returns a list of domains the user has access to.

Inputs:

- <session-id>session-id</session-id>

Returns, for each available domain:

- <domain><domain-id>domain 1 id</domain-id><domain-name>domain 1 name</domain-name></domain>

aspire.chooseDomain

What it does: sets the current domain.

Inputs:

- <session-id>session-id</session-id>
- <domain-id>domain-id</domain-id>

Returns:

- <domain-name>name of the current domain</domain-name>

aspire.listSubdomains

What it does: returns a list of the sub-domains the current domain.

Inputs:

- <session-id>session-id</session-id>

Returns, for each available sub-domain:

- <sub-domain><sub-domain-id>sub-domain 1 id</sub-domain-id><sub-domain-name>sub-domain 1 name</sub-domain-name></sub-domain>

aspire.chooseSubdomain

What it does: sets the current sub-domain.

Inputs:

- <session-id>session-id</session-id>
- <sub-domain-id>sub-domain-id</sub-domain-id>

Returns:

- <sub-domain-id>id of the current sub-domain</sub-domain-id>

aspire.getProblem

What it does: gets the XML for the current problem.

Inputs:

- <session-id>session-id</session-id>

Returns:

- <problem>complete problem structure</problem>

aspire.chooseProblem

What it does: sets the current problem

Inputs:

- <session-id>session-id</session-id>
- <problem-id>problem-id</problem-id>

Returns:

- <problem-id>the problem-id you just set</problem-id>

aspire.selectProblem

What it does: sets the current problem, either the next problem, or the systems choice

Inputs:

- <session-id>session-id</session-id>
- <selection-type>either "system" or "next"</selection-type>

Returns:

- <problem-id>problem-id for the current problem</problem-id>

Note: "system" selection currently doesn't work. This needs investigating.

aspire.getIdealSolution

What it does: gets the ideal solution for the current page of the current problem

Inputs:

- <session-id>session-id</session-id>

Returns:

- <solution>a complete solution structure for the current page</solution>

aspire.recordFullSolutionView

What it does: Records that the user has looked at the full solution

Inputs:

- <session-id>session-id</session-id>

Returns:

- <response>full-solution-recorded</response>

aspire.processAttempt

What it does: evaluates a students solution

Inputs:

- <session-id>session-id</session-id>
- <feedback-level>one of ("pos-neg" "error-flag" "hint" "detailed-hint" "all-errors" "partial-solution" "full-solution")</feedback-level>
- <solution>a student solution (see below)</solution>

Returns:

- <response>(tags: feedback, feedback-type, next-feedback-level,solution-correct-p,whole-page-displayed-p,last-page-p)</response>
- Each feedback tag will (potentially) have a component attribute, listing what component this message was relevant for
- The feedback-type will either be nothing (for a single message) or list (if there are multiple feedback messages)
- Each feedback message may have a set of tags (if highlight-errors is set in the pedagogical strategy). By default, these tags will be instance-ids that can be used to highlight instances with errors in them. Any bindings with ?id, ?ss-id, or ?tag- in them count as tags, so they can be set up by the author when writing the constraints.
- i.e. <response> <feedback component='aRelevantComponent' constraint-id='2_gse'><message>There's something wrong in the aRelevantComponent component</message><tag id='id1' type='instanceid'>3</tag><tag id='tag-sometag' type='general'>somethingbound</tag></feedback><feedback component='anotherone' constraint-id='3_gsy'></feedback><feedback-type>list</feedback-type><next-feedback-level>error-flag</next-feedback-level><solution-correct-p>nil</solution-correct_p><whole-page-displayed-p>t</whole-page-displayed-p><last-page-p>nil</last-page-p></response>

aspire.changePage

What it does: go to the next page for the current problem

Inputs:

- <session-id>session-id</session-id>

Returns:

- <current-page-number>the new page number</current-page-number>

aspire.getCurrentPageNumber

What it does: returns the current page number

Inputs:

- <session-id>session-id</session-id>

Returns:

- <current-page-number>the current page number</current-page-number>

aspire.listSubdomainProblems

What it does: lists the problems in the currently selected subdomain, with id, name, and difficulty

Inputs:

- <session-id>session-id</session-id>

NOTE: domain and subdomain must be set

Returns:
<problems>
<problem><id>id</id><name>name</name><difficulty>diff</difficulty></problem>
<problem>....</problem> (etc)
</problems>

Example

Here is an example of using these RPC calls in LISP

To log in we set up the XML string with username, password and affiliation, and call the login method:

```
(setf args (concatenate 'string "<username>student1</username>"
"<password>student</password>" "<affiliation>ictg</affiliation>"))

(xml-rpc-call
(encode-xml-rpc-call "aspire.login" args)
:url "http://localhost:8080/rpc")
```

This returns us a session-id:

<status>0</status><session-id>265777441898216175775492464332990745619</session-id>

We then use this session-id in all future calls

We can list the domains:

```
(setf session-id "<session-id>265777441898216175775492464332990745619</session-id>")

(xml-rpc-call
(encode-xml-rpc-call "aspire.listDomains" session-id)
:url "http://localhost:8080/rpc")
```

Which returns:
<domain><domain-id>25010</domain-id><domain-name>Capital Investment

Decision</domain-name></domain><domain><domain-id>21011</domain-id><domain-name>Fraction Addition - Demo</domain-name></domain>

We can then choose a domain:

```
(setf domain-id "<domain-id>21011</domain-id>")
(setf args (concatenate 'string session-id domain-id))

(xml-rpc-call
(encode-xml-rpc-call "aspire.chooseDomain" args)
:url "http://localhost:8080/rpc")
```

Returns: <domain-name>Fraction Addition - Demo</domain-name>

Then list the sub domains for the domain we chose:

```
(xml-rpc-call
(encode-xml-rpc-call "aspire.listSubdomains" session-id)
:url "http://localhost:8080/rpc")
```

Returns: <sub-domain><sub-domain-id>0</sub-domain-id><sub-domain-name>set1</sub-domain-name></sub-domain>"

We then choose the sub-domain:

```
(setf sub-domain-id "<sub-domain-id>0</sub-domain-id>")
(setf args (concatenate 'string session-id sub-domain-id))

(xml-rpc-call
(encode-xml-rpc-call "aspire.chooseSubdomain" args)
:url "http://localhost:8080/rpc")
```

Returns: <sub-domain-id>0</sub-domain-id>

We can get the current problem:

```
(xml-rpc-call
(encode-xml-rpc-call "aspire.getProblem" session-id)
:url "http://localhost:8080/rpc")
```

Returns: <problem difficulty='1' name='' id='1'>
<problem-statement statement='1/2 + 1/3'/>
<problem-components/>
<problem-solutions>
<solution id='1' notes='' is-ideal='true'>
<problem-solving-step repeatable='false' id='0'>
<solution-component id='21045' label='LCD'>
<solution-component-instance id='0' concept-name='LCD' free-text='false'>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
<problem-solving-step repeatable='false' id='1'>
<solution-component id='21046' label='Fraction1'>
<solution-component-instance id='1' concept-name='Improper fraction' free-text='false'>
<value-element value='3'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>

<solution-component id='21047' label='Fraction2'>
<solution-component-instance id='2' concept-name='Improper fraction' free-text='false'>
<value-element value='2'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
<problem-solving-step repeatable='false' id='2'>
<solution-component id='21048' label='Sum'>
<solution-component-instance id='3' concept-name='Improper fraction' free-text='false'>
<value-element value='5'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
<problem-solving-step repeatable='false' id='3'>
<solution-component id='21049' label='ReducedSum'>
<solution-component-instance id='4' concept-name='Reduced Fraction' free-text='false'>
<value-element value=''/>
<value-element value=''/>
<value-element value=''/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
</solution>
</problem-solutions>
</problem>

... and the solution:

(xml-rpc-call
(encode-xml-rpc-call "aspire.getIdealSolution" session-id)
:url "http://localhost:8080/rpc")

Returns:
<solution>
<problem-solving-step id='0'>
<solution-component id='21045' label='LCD'>
<solution-component-instance id='0' concept-name='LCD' free-text='false'>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
</solution>

And submit a solution:

(setf solution "
<solution>
<student-solution page-number='1' done='true'>
<problem-solving-step id='0'>
<solution-component id='21045' label='LCD'>
<solution-component-instance id='0' concept-name='LCD' free-text='false'>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
</student-solution>
</solution>")

(setf feedback "<feedback-level>pos-neg</feedback-level>")

(setf args (concatenate 'string session-id solution feedback))

(xml-rpc-call
(encode-xml-rpc-call "aspire.processAttempt" args)
:url "http://localhost:8080/rpc")

Returns: <response><feedback>Well done, you have answered this part of the problem correctly!</feedback><feedback-type></feedback-type><next-feedback-level>error-flag</next-feedback-level><solution-correct-p>t</solution-correct_p><whole-page-displayed-p>t</whole-page-displayed-p><last-page-p>nil</last-page-p></response>"

change page:

(xml-rpc-call
(encode-xml-rpc-call "aspire.changePage" session-id)
:url "http://localhost:8080/rpc")

Returns: <current-page-number>2</current-page-number>

get the ideal solution again:

(xml-rpc-call
(encode-xml-rpc-call "aspire.getIdealSolution" session-id)
:url "http://localhost:8080/rpc")

returns: <solution>
<problem-solving-step id='1'>
<solution-component id='21046' label='Fraction1'>
<solution-component-instance id='1' concept-name='Improper fraction' free-text='false'>
<value-element value='3'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
<solution-component id='21047' label='Fraction2'>
<solution-component-instance id='2' concept-name='Improper fraction' free-text='false'>
<value-element value='2'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>
</solution>

and submit the solution for page 2:

(setf solution "
<solution>
<student-solution page-number='2' done='true'>
<problem-solving-step id='1'>
<solution-component id='21046' label='Fraction1'>
<solution-component-instance id='1' concept-name='Improper fraction' free-text='false'>
<value-element value='3'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
<solution-component id='21047' label='Fraction2'>
<solution-component-instance id='2' concept-name='Improper fraction' free-text='false'>
<value-element value='2'/>
<value-element value='6'/>
</solution-component-instance>
</solution-component>
</problem-solving-step>

</student-solution>
</solution>")

(setf feedback "<feedback-level>all-errors</feedback-level>")

(setf args (concatenate 'string session-id solution feedback))

(xml-rpc-call
(encode-xml-rpc-call "aspire.processAttempt" args)
:url "http://localhost:8080/rpc")

Returns: <response><feedback>Well done, you have answered this part of the problem correctly!</feedback><feedback-type></feedback-type><next-feedback-level>error-flag</next-feedback-level><solution-correct-p>t</solution-correct_p><whole-page-displayed-p>t</whole-page-displayed-p><last-page-p>nil</last-page-p></response>"

Bibliography

[1] J. Gommans, A. Barber, H. McNaughton, C. Hanger, P. Bennett, D. Spriggs, and J. Baskett, "Stroke rehabilitation services in New Zealand," *The New Zealand Medical Journal (Online)*, vol. 116, p. U435, May 2003.

[2] G. J. Hankey and C. P. Warlow, "Treatment and secondary prevention of stroke: Evidence, costs, and effects on individuals and populations," *The Lancet*, vol. 354, pp. 1457–1463, Oct. 1999.

[3] "About Aphasia." http://www.aphasia.org.nz/public/about/public-about, 2010.

[4] R. J. Elman, J. Ogar, and S. H. Elman, "Aphasia: Awareness, advocacy, and activism," *Aphasiology*, vol. 14, no. 5/6, pp. 455–459, 2000.

[5] "Aphasia Information Page: National Institute of Neurological Disorders and Stroke (NINDS)." http://www.ninds.nih.gov/disorders/aphasia/aphasia.htm, 2014.

[6] P. Brown, M. Guy, and J. Broad, "Individual socio-economic status, community socio-economic status and stroke in New Zealand: A case control study," *Social Science and Medicine*, vol. 61, no. 6, pp. 1174–1188, 2005.

[7] K. M. McLellan, C. M. McCann, L. E. Worrall, and M. L. N. Harwood, "Māori experiences of aphasia therapy: "But I'm from Hauiti and we've got shags"," *International Journal of Speech-Language Pathology*, vol. 16, pp. 529–540, Dec. 2013.

[8] C. Code and M. Herrmann, "The relevance of emotional and psychosocial factors in aphasia to rehabilitation.," *Neuropsychological rehabilitation*, vol. 13, no. 1-2, pp. 109–132, 2003.

[9] G. Pearl, "Aphasia." http://www.patient.co.uk/health/aphasia, 2013.

[10] "Aphasia - American Speech-Language Hearing Association." http://www.asha.org/public/speech/disorders/Aphasia/.

[11] A. R. Damasio, "Aphasia," *New England Journal of Medicine*, vol. 326, no. 8, pp. 531–539, 1992. PMID: 1732792.

[12] C. K. Thompson, "Neuroplasticity: evidence from aphasia.," *Journal of communication disorders*, vol. 33, no. 4, pp. 357–66, 2000.

[13] K. Marcotte, D. Adrover-Roig, B. Damien, M. de Préaumont, S. Généreux, M. Hubert, and A. I. Ansaldo, "Therapy-induced neuroplasticity in chronic aphasia," *Neuropsychologia*, vol. 50, no. 8, pp. 1776–1786, 2012.

[14] J. Fridriksson, J. D. Richardson, P. Fillmore, and B. Cai, "Left hemisphere plasticity and aphasia recovery," *NeuroImage*, vol. 60, no. 2, pp. 854–863, 2012.

[15] J. Beck, M. Stern, and E. Haugsjaa, "Applications of AI in education," *Crossroads*, vol. 3, pp. 11–15, 1996.

[16] B. Woolf, "AI in education," in *Encyclopedia of Artificial Intelligence* (S. Shapiro, ed.), pp. 434–444, New York: John Wiley & Sons Inc., 1992.

[17] M. Polson and J. Richardson, *Foundations of Intelligent Tutoring Systems.* Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1988.

[18] S. B. Blessing, S. B. Gilbert, S. Ourada, and S. Ritter, "Authoring model-tracing cognitive tutors," *International Journal of Artificial Intelligence in Education*, vol. 19, no. 2, pp. 189–210, 2009.

[19] S. Ohlsson, "Constraint-based student modeling," *Student modelling: the key to individualized knowledge-based instruction*, pp. 167–189, 1994.

[20] A. Gertner and K. VanLehn, "Andes: A coached problem solving environment for physics," in *Intelligent Tutoring Systems* (G. Gauthier, C. Frasson, and K. VanLehn, eds.), vol. 1839 of *Lecture Notes in Computer Science*, pp. 133–142, Springer Berlin Heidelberg, 2000.

[21] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark, "Intelligent tutoring goes to school in the big city," *International Journal of Artificial Intelligence in Education*, vol. 8, pp. 30–43, 1997.

[22] J. R. Anderson, C. F. Boyle, and G. Yost, "The geometry tutor," in *IJCAI*, pp. 1–7, 1985.

[23] A. Mitrovic, "An intelligent SQL tutor on the web," *International Journal of Artificial Intelligence in Education*, vol. 13, no. 2, pp. 173–197, 2003.

[24] A. Mitrovic, B. Martin, and P. Suraweera, "Intelligent tutors for all: The constraint-based approach," *IEEE Intelligent Systems*, no. 4, pp. 38–45, 2007.

[25] G. Westerfield, A. Mitrovic, and M. Billinghurst, "Intelligent augmented reality training for motherboard assembly," *International Journal of Artificial Intelligence in Education*, vol. 25, no. 1, pp. 157–172, 2015.

[26] Y. Oh, M. Gross, S. Ishizaki, and Y. Do, "Constraint-based design critic for flat-pack furniture design," in *Proc. 17th International Conference of Computers in Education* (S. Kong, H. Ogata, H. Arnseth, C. Chan, T. Hirashima, F. Klett, J. Lee, C. Liu, C. Looi, M. Milrad, A. Mitrovic, K. Nakabayashi, S. Wong, and S. Yang, eds.), pp. 19–26, 2009.

[27] J. Greer, G. McCalla, and N. A. T. O. S. A. Division, *Student Modelling: The Key to Individualized Knowledge-based Instruction*. Lecture Notes in Computer Science, Springer-Verlag, 1994.

[28] A. Mitrovic, "NORMIT, a web-enabled tutor for database normalization," University of Canterbury. Computer Science and Software Engineering, 2002.

[29] A. Mitrovic, B. Martin, P. Suraweera, K. Zakharov, N. Milik, J. Holland, and N. McGuigan, "ASPIRE: an authoring system and deployment environment for constraint-based tutors," *International Journal of Artificial Intelligence in Education*, vol. 19, no. 2, pp. 155–188, 2009.

[30] K. Koedinger, V. Aleven, N. Heffernan, B. McLaren, and M. Hockenberry, "Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration," in *Intelligent Tutoring Systems* (J. Lester, R. Vicari, and F. Paraguaçu, eds.), vol. 3220 of *Lecture Notes in Computer Science*, pp. 162–174, Springer Berlin Heidelberg, 2004.

[31] A. Mitrovic, M. Mathews, S. Ohlsson, J. Holland, A. McKinlay, S. Ogden, A. Bracegirdle, and S. Dopping-Hepenstal, "From tutoring to cognitive rehabilitation: Exploiting CBM to support memory training," pp. 32–41, 2014.

[32] A. Mitrovic, M. Mathews, S. Ohlsson, J. Holland, A. McKinlay, S. Ogden, A. Bracegirdle, and S. Dopping-Hepenstal, "A virtual reality environment for prospective memory training," in *Proceedings of UMAP 2014 posters, demonstrations and late-breaking results* (C. M. Cantador, I, ed.), pp. 93–98, 2014.

[33] R. Wertz and R. Katz, "Outcomes of computerprovided treatment for aphasia," *Aphasiology*, vol. 18, no. 3, pp. 229–244, 2004.

[34] C.-W. Wallesch and H. Johannsen-Horbach, "Computers in aphasia therapy: Effects and sideeffects," *Aphasiology*, vol. 18, no. 3, pp. 223–228, 2004.

[35] H. Guyard, V. Masson, and R. Quiniou, "Computer-based aphasia treatment meets artificial intelligence," *Aphasiology*, vol. 4, no. 6, pp. 599–613, 1990.